Обработка исключительных ситуаций

Типы ошибок, которые могут возникать в программах

В программах на C++ могут возникать ошибки. Различают три типа ошибок, которые могут возникать в программах:

- синтаксические. Это ошибки в синтаксисе языка C++. Они могут встречаться в именах операторов, функций, разделителей и т.д. В этом случае компилятор определяет наличие синтаксической ошибки и выдает соответствующее сообщение. В результате исполняющий (*.exe) файл не создается, и программа не выполняется;
- логические. Это ошибки программиста, которые сложно обнаружить на этапе разработки программы. Эти ошибки обнаруживаются на этапе выполнения во время тестирования работы программы. Логические ошибки можно обнаружить только по результатам работы программы. Примером логических ошибок может быть неправильная работа с указателями в случаях выделения/освобождения памяти;
- ошибки времени выполнения. Такие ошибки возникают во время работы программы. Ошибки времени выполнения могут быть логическими ошибками программиста, ошибками внешних событий (например, нехватка оперативной памяти), неверным вводом данных пользователем и т.п. В результате возникновения ошибки времени выполнения, программа приостанавливает свою работу. Поэтому, важно перехватить эту ошибку и правильно обработать ее для того, чтобы программа продолжила свою работу без остановки.

Понятие исключительной ситуации

Исключительная ситуация – это событие, которое привело к сбою в работе программы. В результате возникновения исключительной ситуации программа не может корректно продолжить свое выполнение.

Примеры действий в программе, которые могут привести к возникновению исключительных ситуаций:

деление на нуль;

нехватка оперативной памяти при использовании оператора new для ее выделения (или другой функции);

- доступ к элементу массива за его пределами (ошибочный индекс);
- переполнение значения для некоторого типа;
- взятие корня из отрицательного числа;
- другие ситуации.

Понятие исключения

В языке C++ *исключение* – это специальный объект класса или значение базового типа, который описывает (определяет) конкретную исключительную ситуацию и соответствующим образом обрабатывается.

При написании программы система описания исключительных ситуаций выбирается программистом по собственному усмотрению. Можно создать свою квалификацию ошибок, которые могут возникать в программе. Например, программист может квалифицировать разные типы ошибок числовым (целочисленным) значением или разработать собственную иерархию классов описывающих исключительные ситуации. Кроме того, можно использовать возможности классов C++, которые являются производными от класса exception.

Средства языка C++ для обработки исключительных ситуаций. Общая форма конструкции try…catch.

Язык программирования C++ дает возможность перехватывать исключительные ситуации и соответствующим образом их обрабатывать.

Механизм перехвата исключений C++ позволяет генерировать исключения в том месте, в котором оно возникает – это очень удобно. Не нужно «выдумывать» собственные способы обработки исключений, которые возникают в функциях нижних уровней, для того чтобы передать их в функции высших уровней.

Для перехвата и обработки исключительных ситуаций в языке C++ введена конструкция try…catch, которая имеет следующую общую форму:

```
try {
  // тело блока try
  // ...
  // генерирование исключения оператором throw
}
catch(type1 argument1)
{
  // тело блока catch
}
catch(type2 argument2)
{
  // тело блока catch
}
...
catch(typeN argumentN)
{
  // тело блока catch
}
```

где

- *type1*, *type2*, …, *typeN* – соответственно тип аргументов *argument1*, *argument2*, …, *argumentN*.

Код, который нужно проконтролировать, должен выполняться всередине блока try. Исключительные ситуации перехватываются оператором catch, который следует непосредственно за блоком try в котором они возникли.

В блоке try могут быть размещены операторы и функции. Если в блоке try генерируется соответствующая исключительная ситуация, то она перехватывается соответствующим блоком catch. Выбор того или иного блока catch осуществляется в зависимости от типа исключительной ситуации. После возникновения исключительной ситуации определенного типа, вызывается блок catch с таким самым типом аргумента. Аргумент принимает некоторое значение, которое соответствующим образом обрабатывается (выводится на экран сообщение об ошибке и т.п.).

Если в блоке try возникнет исключительная ситуация, которая не предусмотрена блоком catch, то вызывается стандартная функция terminate(), которая по умолчанию вызовет функцию abort(). Эта стандартная функция останавливает выполнение программы.

Оператор throw.

Чтобы в блоке try сгенерировать исключительную ситуацию, нужно использовать оператор throw. Оператор throw может быть вызван внутри блока try или внутри функции, которая вызывается из блока try.

Общая форма оператора throw следующая

throw исключение;

В результате выполнения оператора throw генерируется исключение некоторого типа. Это исключение должно быть обработано в блоке catch.

Примеры использования блока try…catch

Пример 1. Демонстрируется использование блока try…catch для обработки выражения:

- В данном выражении в трех случаях может возникнуть исключительная ситуация:
- корень из отрицательного числа a, если a<0;
- корень из отрицательного числа b, если b<0;
- деление на 0, если b=0.

Поэтому, в блоке try…catch нужно обработать эти три случая.

Текст программы типа Console Application следующий:

```
#include <iostream>
using namespace std;

void main()
{
 // обработка выражения sqrt(a)/sqrt(b)
 double a, b;
 cout << "a = ";
 cin >> a;

 cout << "b = ";
 cin >> b;

 double c;

 try { // начало блока try
  if (b == 0)
    throw 1;
  if (b < 0)
    throw 2;
  if (a < 0)
    throw 2;

  c = sqrt(a) / sqrt(b);
  cout << "c = " << c << endl;
 }
 catch (int e) // перехват ошибки
 {
  if (e == 1)
    cout << "Division by 0." << endl;
  if (e == 2)
    cout << "Negative root." << endl;
 }
}
```

Результат работы программы
a = 5
b = 0
Division by 0.

После применения блока try..catch работа программы не прекращается.

Пример 2. Другой вариант обработки выражения из примера 1. Здесь блок try…catch содержит два оператора catch.

```
#include <iostream>
using namespace std;

void main()
{
 // обработка выражения sqrt(a)/sqrt(b) - вариант 2
 double a, b;
 cout << "a = ";
 cin >> a;

 cout << "b = ";
 cin >> b;

 double c;
 string s;

 try { // начало блока try
  if (b == 0)
   throw "Division by 0.";
  if (b < 0)
   throw "Negative root.";
  if (a < 0)
   throw "Negative root.";

  // если исключительных ситуаций нет, то продолжить вычисления
  c = sqrt(a) / sqrt(b);
  cout << "c = " << c << endl;
 }
 catch (int e) // перехват ошибки типа int
 {
  if (e == 1)
   cout << "Division by 0." << endl;
  if (e == 2)
   cout << "Negative root." << endl;
 }
 catch (const char* e) // перехват ошибки типа const char*
 {
  cout << e << endl;
 }
}
```

Пример использования блока try…catch в функции

Условие задачи. Написать функцию вычисления значения по заданной строке символов, которая есть записью этого числа в десятичной системе исчисления. Предусмотреть случай выхода за границы диапазона, определяемого типом int. Необходимо использовать механизм исключений.

Текст программы для приложения типа Console Application следующий

```
#include <iostream>
using namespace std;

// Функция вычисления значения по заданной строке символов
int StrToInt(const char* str)
{
  char s[20];
  int t, i;
  long res = 0; // результат возврата из функции
  int len = strlen(str);

  try
  {
    t = 1;
    if (str[0] == '-')
      t = -1; // проверка, первый ли символ '-'

    // цикл конвертирования строки в число типа int
    i = len - 1;
    while (i >= 0)
    {
      if (str[i] == '-')
      {
        if (i == 0) break; // если перший символ '-', то все в порядке
        else throw "Bad position of minus.";
      }

      // если в строке недопустимые символы, то сгенерировать исключение
      if (str[i] < '0') throw "Bad symbols";
      if (str[i] > '9') throw "Bad symbols";

      res = res + (str[i] - '0')*t;

      t *= 10;
      i--;
    }

    // если результат выходит за пределы диапазона для 32-разрядных
    // целочисленных значений, то сгенерировать соответствующее исключения
    if (res > INT32_MAX)
      throw "Out of range.";
    if (res < INT32_MIN)
      throw "Out of range.";
    return res;
  }
```

```
 catch (const char* e)
 {
  cout << e << endl;
  return 0;
 }
}

void main()
{
 int d;
 d = StrToInt("125");
 cout << "d = " << d;
}
```

Вышеприведенная программа может быть переписана так, что блок try…catch размещается в функции main(), как показано ниже

```
#include <iostream>

using namespace std;

// Функция вычисления значения по заданной строке символов
int StrToInt2(const char* str)
{
 char s[20];
 int t, i;
 long res = 0; // результат работы функции
 int len = strlen(str);

 t = 1;
 if (str[0] == '-') t = -1;

 i = len - 1;
 while (i >= 0)
 {
  if (str[i] == '-')
  {
   if (i == 0) break; // если первый символ '-', то все в порядке
   else throw "Bad position of minus."; // иначе, сгенерировать исключение
  }

  if (str[i] < '0') throw "Bad symbols";
  if (str[i] > '9') throw "Bad symbols";

  res = res + (str[i] - '0')*t;
  t *= 10;
  i--;
 }

 // если результат выходит за пределы диапазона для 32-разрядных
 // целочисленных значений, то сгенерировать соответствующее исключения
 if (res > INT32_MAX)
  throw "Out of range.";
 if (res < INT32_MIN)
```

```
   throw "Out of range.";
  return res;
}

void main()
{
  int d;

  // блок try...catch размещается в функции main() высшего уровня,
  // а исключение генерируется в функции StrToInt2() нижнего уровня
  try {
    d = StrToInt2("19125");
    cout << "d = " << d;
  }
  catch (const char* e)
  {
    cout << e << endl;
  }
}
```

Как видно из вышеприведенного кода, генерировать исключения оператором throw можно в другой функции, вызов которой включен в блок try. Значит, функция в своем теле может генерировать исключение.

Результат выполнения программы

```
d = 19125
```

Если вызов функции StrToInt2() перенести за пределы оператора try

```
void main()
{
  int d;

  try {
    //d = StrToInt2("19125");
    //cout << "d = " << d;
  }
  catch (const char* e)
  {
    cout << e << endl;
  }

  // вызов функции за пределами оператора try
  d = StrToInt2("в19125");
}
```

то исключительные ситуации в функции StrToInt2() обрабатываться не будут. При возникновении исключительной ситуации в функции StrToInt2() компилятор сгенерирует собственную ошибку

```
Exception Unhandled
```

что означает, что исключение необработано.

Пример программы, которая генерирует исключение в одной функции, а перехватывает его в другой функции

В примере, в функции нижнего уровня GenerateException() генерируется исключение типа const char*. Функция проверяет допустимые границы входного параметра index.

В функции верхнего уровня ProcessException() происходит вызов функции GenerateException(). Этот вызов взят в блок try.

Текст программы следующий:

```
#include <iostream>
using namespace std;

// Пример Программы, которая генерирует исключение в одной функции, а перехватывает его в другой
// Функция генерирует исключение "Out of index",
// если значение index находится за пределами диапазона 0..9
void GenerateException(int index)
{
  if ((index < 0) || (index > 9))
    throw "Out of index";
}

// Функция, которая перехватывает исключение "Out of index"
void ProcessException()
{
  int index;
  cout << "index = ";
  cin >> index;

  // 1. Вызвать исключительную ситуацию без обработки,
  // компилятор выдаст сообщение "Exception unhandled"
  // GenerateException(-3);

  // 2. Вызвать исключительную ситуацию с обработкой блоком try...catch
  try {
    GenerateException(index); // вызов функции, которая генерирует исключение
    cout << "OK!" << endl; // если index в пределах 0..9, то OK!
  }
  catch (const char* e)
  {
    cout << "Error: " << e << endl;
  }
}

void main()
{
  ProcessException();
}
```

Результат выполнения программы

```
index = -5
Error: Out of index
```

Использование блока catch(…). Перехват всех возможных исключительных ситуаций.

Бывают случаи, когда нужно перехватить все исключительные ситуации подряд. Для этого, в C++ используется блок catch(…), который имеет следующую общую форму
`catch(...)`

```
{
  // Обработка всех исключительных ситуаций
  // ...
}
```

Пример. В примере демонстрируется использование блока catch(…) для обработки ситуаций любого типа.

В программе реализованы:

- функция DivNumbers(), которая возвращает результат деления двух чисел, введенных из клавиатуры. В функции генерируется исключение типа int, если значение делителя равно 0;
- функция SqRoot(), возвращающая корень из отрицательного числа. В функции генерируется исключение типа const char*, если значение параметра number отрицательно;
- функция ProcessException(). Эта функция демонстрирует работу функций DivNumbers() и SqRoot(). В функции используется инструкция try…catch().

```
#include <iostream>
using namespace std;

// Пример. Демонстрация использования блока catch
// Функция, которая делит 2 числа и возвращает результат
double DivNumbers(double a, double b)
{
  if (b == 0) throw 1;
    return a / b;
}

// Функция, возвращающая корень из отрицательного числа
double SqRoot(double number)
{
  if (number < 0) throw "Negative number";
    return sqrt(number);
}

// Демонстрация блока catch(...)
void ProcessException()
{
  double v;

  // цикл отображения и вызова нужной функции
  while (1)
  {
    cout << "Input a function to call (1-2, 3-exit): " << endl;
    cout << "1-DivNumbers. 2-SqRoot" << endl;
    cout << ">>";
    cin >> v;
```

```
   // Вызвать различные варианты функций
   try {
     if (v == 1) // функция DivNumbers
     {
       double a, b;
       cout << "DivNumbers(double a, double b)" << endl;

       // ввести a, b
       cout << "a = "; cin >> a;
       cout << "b = "; cin >> b;

       // Вызвать функцию DivNumbers()
       double c = DivNumbers(a, b);
       cout << "c = " << c << endl;
     }
     if (v == 2)
     {
       double x, num;
       cout << "SqRoot(double num)" << endl;
       cout << "num = "; cin >> num;

       // Вызвать функцию SqRoot()
       x = SqRoot(num);
       cout << "x = " << x << endl;
     }
     if (v == 3) break;
   }
   catch (const char* e)
   {
     cout << "Error. Text = " << e << endl;
   }
   catch (...) // все другие типы исключений
   {
     cout << "Error in block catch(...)." << endl;
   }
  }
}

void main()
{
  ProcessException();
}
```

Как видно из текста функции ProcessException() вызов функций DivNumbers() и SqRoot() взят в блок try…catch

```
// Вызвать разные варианты функций
try {
  ...
}
catch (const char* e)
{
  cout << "Error. Text = " << e << endl;
```

```
}
catch (...) // все другие типы исключений
{
  cout << "Error in block catch(...)." << endl;
}
```

В блоке try…catch обрабатываются

- исключение типа const char*;
- все другие типы исключений. В этом случае используется инструкция catch(…).

Результат работы программы

```
Input a function to call (1-2, 3-exit):
1-DivNumbers. 2-SqRoot
>>2
SqRoot(double num)
num = -4
Error. Text = Negative number
Input a function to call (1-2, 3-exit):
1-DivNumbers. 2-SqRoot
>>1
DivNumbers(double a, double b)
a = 3
b = 0
Error in block catch(...).
Input a function to call (1-2, 3-exit):
1-DivNumbers. 2-SqRoot
>>1
DivNumbers(double a, double b)
a = 2
b = 5
c = 0.4
Input a function to call (1-2, 3-exit):
1-DivNumbers. 2-SqRoot
>>3
```